A VECTORIZED LU DECOMPOSITION ALGORITHM FOR LARGE SCALE CIRCUIT SIMULATION(Numerical Algorithms of Large Linear Problems)

AUTHOR(S):

Yamamoto, Fujio; Takahashi, Sakae

数理解析研究所講究録
548巻 1985年 17-29

A VECTORIZED LU DECOMPOSITION ALGORITHM FOR LARGE SCALE CIRCUIT SIMULATION

Fujio Yamamoto and Sakae Takahashi
山本富士男　　高橋　栄
Central Research Laboratory, Hitachi Ltd., Kokubunji, Tokyo 185

Abstract

A new LU decomposition algorithm that is well-suited for a vector processor with the indirectly indexed vector feature is presented. Application of this algorithm to large scale circuit simulation has reduced the simulation time by an order of magnitude.

Introduction

Conventional circuit simulators such as SPICE2[1] have been widely used because of their general applicability and high accuracy. Much efforts have been made to successfully improve their performance on general-purpose computers, as typified by the code generation scheme[2]. Rapid growth in circuit integration in this VLSI era, however, requires a circuit simulator with much higher performance.

One approach to this target is the relaxation method such as those adopted in RELAX2[3], SPLICE1[4] and New MOTIS[5]. One problem with this approach lies in deterioration of simulation accuracy that comes out explicit when applied to analog circuits, and research works are still under way to overcome or mitigate this weak point.

Another approach is to accelerate a conventional simulator, while preserving its wide applicability and high accuracy, by the

use of a vector processor such as CRAY-1, Fujitsu VP or Hitachi S-810. The point here is to develop a new computational algorithm for circuit simulation that is well-suited for vectorized processing. In his comprehensive position paper[6], Calahan stated that matrix computation that becomes more dominant in larger scale simulation, is hard to vectorize efficiently, due to high sparsity and randomness inherent in circuit matrices. Afterwards, he[7] and others[8] presented the vector processing schemes that are effective only for circuits consisting of repeated sub-circuits with the same topology.

This paper presents a new algorithm of sparse matrix computation that is well-suited for a vector processor with indirectly indexed vector feature, and is effective for general circuits that do not necessarily explicate topological regularity. As a matter of fact, this algorithm is based on the parallelism latent in high sparsity and irregularity of circuit matrices, the very nature that has been regarded as the major obstacle against vector acceleration.

Towards Acceleration of Conventional Simulators

One of our VLSI circuits (benchmark A in Table 1) that is modeled with 2,132 MOS transistors,takes 4.2 hours to simulate on our high-end scalar processor. Nearly 90% of the total time is spent for matrix computation,of which over 90% is for the LU decomposition process. Characteristics of a circuit matrix stated above make it difficult to apply those solution methods that are commonly used for PDE problems,such as the iterative method or various band matrix schemes. Consequently, traditional LU decomposition enhanced with fill-in minimization

pre-processing has been widely adopted.

The code generation scheme[2], though based on the direct method as well, provides with much higher performance. The large memory capacity required by this scheme, however, prevents it from freely used for VLSI chip simulation. We gave it up to apply this to our benchmark A due to memory shortage, since it required the code memory capacity of 22MB, the amount much larger than what was available.

On the other hand, the traditional LU decomposition process, as it stands, can be vectorized rather easily and naturally on a vector processor equipped with the indirectly indexed vector feature. For example, you can easily vectorize each set of divide-operations and update-operations that are related to each diagonal element. Such a "naive" vectorization, however, will not give you much, since high sparsity of a circuit matrix cuts the vector length quite short. Worse yet, the relatively simple and short sequence of floating-point operations that is peculiar to LU decomposition loop, is apt to lower the utilization ratio of vector arithmetic hardware. Another vectorized LU decomposition algorithm that aims at overcoming these flaws will be given in following.

Outline of the Algorithm

The vectorized LU decomposition algorithm proposed here preprocesses a given matrix in the following three steps, thereby examining the potentiality for parallel processing and preparing for vectorization.

(1) Segment the whole matrix into a number of "blocks", so that,

in each block, a set of divide-operations (Aji=Aji/Aii) and a set of update-operations (Ajk=Ajk-Aji*Aik) can be vectorized respectively. Fig.1 illustrates a primitive example of this segmentation. As shown, each block consists of some consecutive columns and the corresponding rows. Each block boundary is uniquely determined as the trade-off between two requirements. Vector acceleration requires the vector length,and so the block size,be as large as possible. Meanwhile, computational integrity requires that each block does not include an element which is written by the vector operations of that block, except those that are, after written, not referred by these operations. This step ends with building index lists for vector operations of each block.

(2) Detect those matrix elements for which two or more update-operations will be undertaken while processing a block. Such an element is located at Ajk when there are two or more pairs of non-zero Aji and Aik for common values of j and k, as shown by solid circles in Fig.1. These "singular" elements should be set aside in the course of vectorized update-operations stated in (1), to avoid unexpected results caused by concurrent updates. Instead, these elements are grouped according to the number of update-operations taken, so that another vector operation of the form of (Ajk=Ajk-Aji*Aik-Aji'*Ai'k-...) can be applied, later in execution, to each group having many multiple-update elements enough to justify vectorization. A list of operand address is appended to each of multiple-update elements (Ajk).

(3) Determine the point along the diagonal where the computational method for a regular dense matrix should be adopted to decompose the remaining lower-right corner. The vector operations mentioned in (1) and (2) assume the indirectly indexed vector feature that enables the usage of a non-linear index for a sparse matrix. The lower-right corner of a matrix, however, tends to get dense due to the preceding update-operations. This implies the advantage of switching to the ordinary vector operation with linear index for a dense matrix. This step fixes the point along the diagonal where the remaining lower-right corner reaches certain sparsity level and so can be advantageously regarded as a dense matrix. Block generation mentioned in (1) is stopped at this point.

These preparation steps are followed by execution of actual decomposition, where, as shown in Fig.1, one divide-operation, one update-operation for normal elements,and some number (possibly zero) of update-operations for multiple-update elements, are taken in order, in the vector mode, for each block. Vectorized execution of the last type of operation in the form stated in (2) improves the utilization of vector arithmetic hardware effectively.

After all the blocks undergo these types of operation, the lower-right corner yet undecomposed is processed by a high-speed decomposition method for a dense matrix. The main loop in this method contains the floating-point operations four times more than the normal update-loop, hence keeping the vector arithmetic

efficiency much higher.

Results

Table1 illustrates some of the performance results measured on our supercomputer S-810. Here, the original version (which is based on UCB's SPICE2) and the code generation version are executed in the scalar mode, while the new version that incorporates the algorithm just given is executed in the vector mode. As for benchmark A that has the largest integration, the new version is 76 times faster in matrix computation and 8.9 times faster in total, as compared with the original. Moreover, it exhibits the superiority to the code generation by a factor of 4.6 in matrix computation, or 1.8 in total, with only half the memory occupation.

In the case of other benchmark having mediumscale, the ratio of the new to the original is 1.3-2.4 in the total time, but that to the code generation stays around 0.7-1.1.

Analysis of Benchmark A

The results with most favorable benchmark are analyzed in three ways, as shown in Fig.2.

Fig.2(a) tells you how the amount of update/divide operation and the number of the vector operation activations accumulate as the LU decomposition using non-linear indices proceeds. (Decomposition of the lower-right corner is not included here.) You can see the new algorithm makes the average vector length, which is implied by ratio of two amounts indicated, about 20 times longer than the "naive" vectorization such as the one

mentioned earlier. In this case, the naive vectorization was realized by a small modification on the original version that limits the scope of vectorization to each column or each row.

Fig.2(b) illustrates the effect of switching, in the course of decomposition process, to the dense matrix method. X-axis represents the size of the lower-right corner that is regarded as dense, while the right-hand Y-axis indicates the amount of operation needed to decompose the corner. The solid curve and the dashed line correspond to the sparse matrix method and the dense matrix one, respectively. The former uses indirect indices for non-zero elements, but the latter operates on all the elements regardless of their values. Also plotted in the figure, against the left-hand Y-axis, is the time it takes to process the whole matrix for one iteration.

Notice that the optimal point of switching is located somewhat before the point where the remaining corner gets completely dense. In this example, the maximum performance is attained when the switching takes place at the point where the amount of operation as a dense submatrix roughly doubles that as a sparse one. The performance value at this optimal point is 1.3 times greater than when no switching is sdopted, and 1.2 times greater than when switching takes place at the completely dense point.

How several factors contribute to the performance effect that matrix computation itself is accelerated 76 times by the new algorithm, is summarized in Fig.2(c). As shown, the effect of removing iterative address search for non-zero elements from the inner execution loop and substituting for it the address lists that are generated once in preprocessing and used repeatedly in

execution, is almost one order of magnitude. There are also the effect of replacing scalar operations with vectorized ones both for sparse and dense submatrices, and that of decreasing load and store operations for multiple-update elements. Contributions of vector elongation represented in Fig.2(a) and dynamic switching in Fig.2(b) are included in these two factors in Fig.2(c).

Matrix characteristics vs performance effect

Table2 summarizes major characteristics of the matrices used to model benchmark circuits.

First, take a look at the third row. The value in each column indicates the ratio of the total number of circuit components to the matrix size, which roughly represents the average number of components connected to one circuit node. Benchmark D, the least favorable one, has the value of only 0.67, which differs much from 3.76 of benchmark A which exhibits the highest acceleration. The low value of benchmark D is due to its bipolar transistor model which holds several internal nodes.

As implied earlier in the description of step(2), a circuit matrix first undergoes node renumbering based on Markowitz's scheme prior to the three-step preprocessing. The small number of components per node with benchmark D should suggest fewer fill-ins after renumbering. Actually, the fill-in ratio in table2 is only 49% with D, much lower than 117% of A.

Markowitz's scheme generally results in higher density at the right hand side, the lower side and the lower-right corner of a matrix. However, the low value of the fill-in ratio weaken this tendency. This comes into sight when we examine the distribution of non-zero elements,which is shown also in Table2. You can see

benchmark D has the distribution that is more balanced than others. This implies it has pretty many non-zeros near the diagonal, that makes our block segmentation feature hash the whole matrix into many small blocks, thus falling down the resultant performance. The average size of a block, which is measured by the number of columns per block, is only 1.8 with D, much smaller than others as indicated in the table.

In the case of our benchmark D,worse yet, the size of the lower-right corner to which the dense matrix method is applied is very small, so that the effect of dynamic switching is quite low.

Conclusion

A new LU decomposition algorithm that effectively brings out high performance of a vector processor, and its effects on large-scale circuit simulation, have been presented. It has been observed that this algorithm contribute to the total acceleration of conventional circuit simulatior by one order of magnitude at the maximum.

On the other hand, it has turned out that some circuits are unamenable to significant acceleration by this algorithm. It is our temporal conclusion that a circuit matrix which holds many non-zeros along the diagonal after the fill-in minimization process should not conform to this algorithm.

Acknowledgment

The skillful method to decompose the dense submatrix that makes a significant part of the algorithm presented in this paper, is attributed to Mr.Yasunori Ushiro. Dr.Kiyoo Itoo, Dr.Hisashi Horikoshi and Dr.Hideki Fukuda have supported this

work eagerly. The authors are much grateful especially to these persons among many others.

References

[1]L.W.Nagel,"SPICE2- A Computer Program to simulate Semiconductor Circuits", MEMO NO.ERL-M520,University of California,Berkeley,1975

[2]F.G.Gustavson,et al, "Symbolic generation of an Optimal Crout Algorithm for Sparse Systems of Linear Equations", Proc.Symposium on Sparse Matrices, pp.1-10,1968

[3]J.White,Sangiovanni-Vicentelli,"RELAX2:A Modified Waveform Relaxation Approach to the Simulation of MOS Digital Circuits", Proc. ISCAS'83, pp.756-759,1983

[4]R.A.Saleh,A.R.Newton,"Iterated Timing Analysis in SPLICE1", Proc.ICCAD'83, pp.139-140,1983

[5]C.F.Chen,P.Subramanian,"The Second Generation MOTiS Timing Simulator-An Efficient and Accurate Approach for General MOS Circuits", Proc. ISCAS'84,pp.538-542,1984

[6]D.A.Calahan, "Vector processors-models and Applications", Trans. on CAS No.9, pp.715-726,1979

[7]D.A.Calahan,"Multi-Level Vectorized Sparse Solution of LSI Circuits",Proc.ICCC'80, pp.976-979, 1980

[8]A.Vladimirescu,D.O.Pederson, "Circuit Simulation on Vector Processors", Proc.ICCC'82, pp172-175, 1982

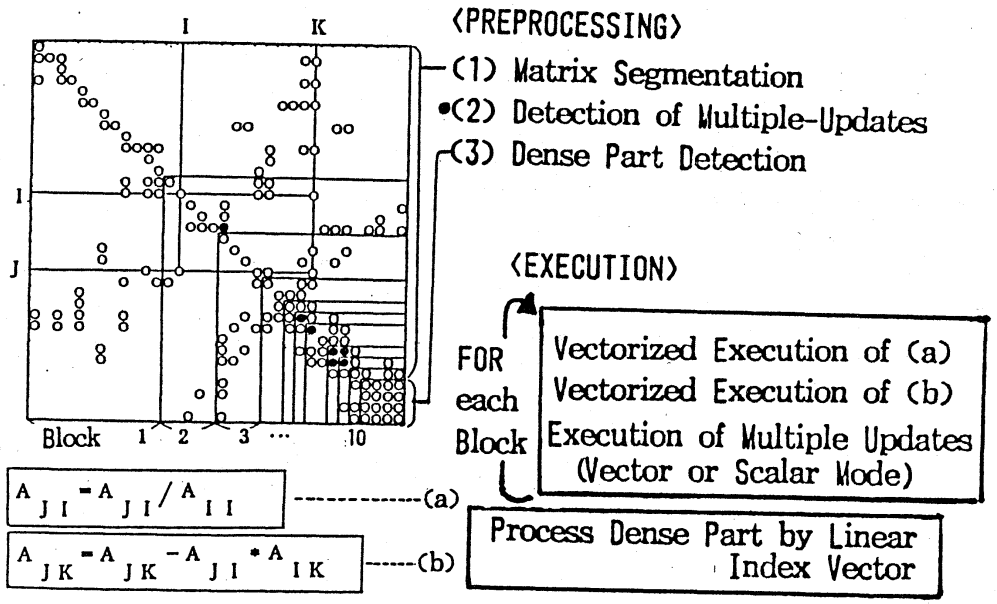
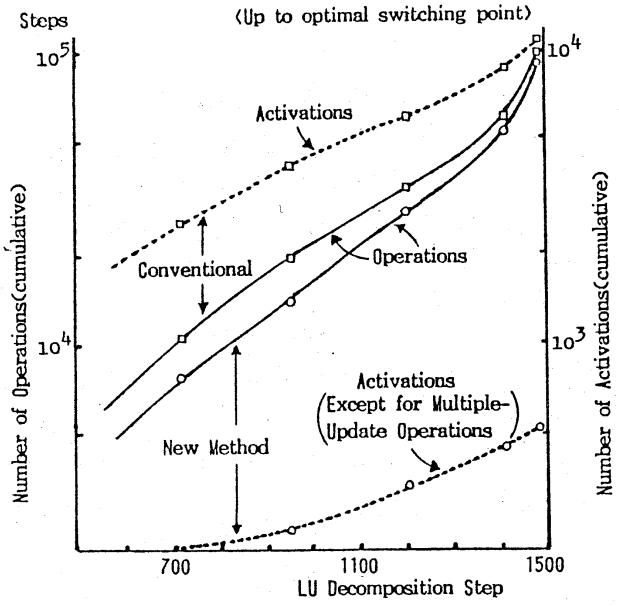


Fig. 1 Principle of the New LU Decomposition Method

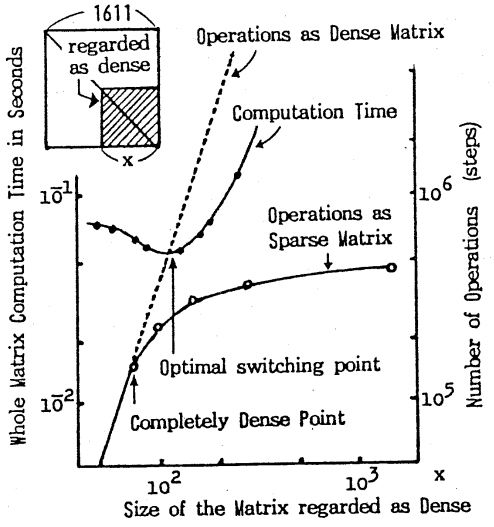
Table 1 Comparison of Simulation Time(CPU Time in Seconds on S810)

Benchmark Identifier		A	B	C	D
Circuit Size (Number of Transistors)		2132 MOST	231 MOST	791 MOST	471 BJT
Conventional Method	Total	1) 14950	764	1137	1061
	Matrix Only	13026	498	654	585
Code Generation	Total	2) 3000	341	527	586
	Matrix Only	790	75	85	140
New Method	Total	3) 1674	314	554	836
	Matrix Only	172	45	95	369

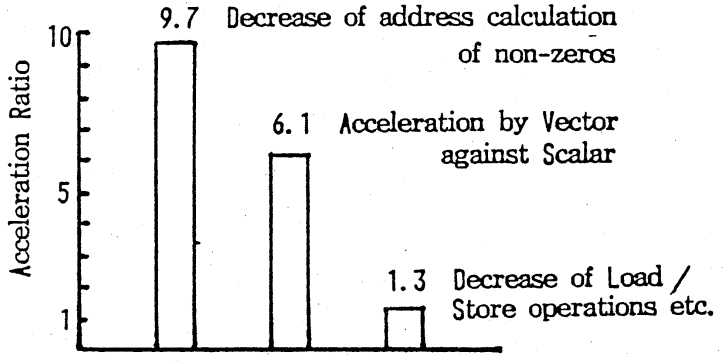
Main Memory Occupation: 1)6MB, 2)22MB, 3)12MB



(a) Comparison of Vector Operation Activations



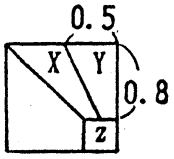
(b) Effect of Switching to Dense Matrix Method



(c) Factors contribute to 76-fold Acceleration

Fig. 2 Analysis of Benchmark A

Table 2 Characteristics of Circuit Matrices

Benchmark Identifier		A	B	C	D
Matrix Size		1611	369	985	1883
Fill-in Ratio (%)		117	62	52	49
Total NUmber of circuit Components/Matrix Size		3.76	3.02	1.98	0.67
Number of Columns/Block		5.8	7.8	3.5	1.8
Distribution of Non-Zeros (%) 	Region X	5.2	6.0	5.8	15.9
	Region Y	16.0	20.5	25.9	18.3
	Region z	57.4	45.6	36.4	31.5